

# Analiza podataka i regresija

**Analiza podataka** je proces inspekcije, filtriranja, transformacije i modeliranja podataka u cilju formiranja korisnih informacija, donošenja informisanih odluka i podrške u odlučivanju. Prva faza procesa analize podataka zahteva njihovo prikupljanje. Zatim se prikupljeni podaci modeliraju, odnosno prilagođavaju informacionom sistemu u skladu sa situacijama iz realnog sveta. Iz ovako modeliranih podataka moguće je donošenje zaključaka i utvrđivanje anomalija. Ključna uloga analize podataka je pretvaranje podataka u informacije.

Za analizu podataka možemo koristiti dve grupe alata:

- **alati za automatsko upravljanje** – Excel, Tableau, Looker...;
- **programski jezici** – Python, R, Julia...

Kada vršimo odabir alata za analizu podataka, od izuzetne je važnosti da sagledamo sve nedostatke i pogodnosti koje nam ti alati pružaju (tabela 4.1).

Karakteristike alata za analizu podataka	
Alati za automatsko upravljanje	Programski jezici
zatvoren kod	otvoren kod
skupi	besplatni ili vrlo jeftini
ograničeni	izuzetno moćni
laki za upotrebu	teži za upotrebu

Tabela 4.1. Poređenje karakteristika alata za analizu podataka

Zbog svoje jednostavnosti, intuitivnosti, čitljivosti i velikog broja biblioteka koje nude različite mogućnosti, Python predstavlja najbolji jezik za kodiranje. Njegova javna dostupnost je još jedna od prednosti koja ga visoko kotira pri izboru alata za analizu.

Analiza podataka odvija se u nekoliko ključnih faza:

- **prikupljanje podataka** – iz fajlova (CSV, JSON i sl.), distribuiranih baza podataka;
- **prečišćavanje podataka** – nedostajuće vrednosti, prazni podaci, nevalidni tipovi i vrednosti;
- **oblikovanje podataka** – sortiranje, klasteriranje, indeksiranje, povezivanje itd.;
- **proučavanje podataka** – istraživanje, statističke analize, hipoteze, vizualizacija;
- **aktivnosti** – izrada modela za mašinsko učenje, kreiranje izveštaja, inženjering, ETL procesi itd.

Postoji više biblioteka za analizu podataka koje Python koristi. Kao najčešće upotrebljavane izdvajamo:

- **pandas** – biblioteka na kojoj se temelji struktura analiza podataka u Pythonu;
- **Matplotlib** – biblioteka na kojoj se temelji vizualizacija podataka;
- **NumPy** – numerička biblioteka na kojoj se baziraju n-dimenzionalni nizovi;

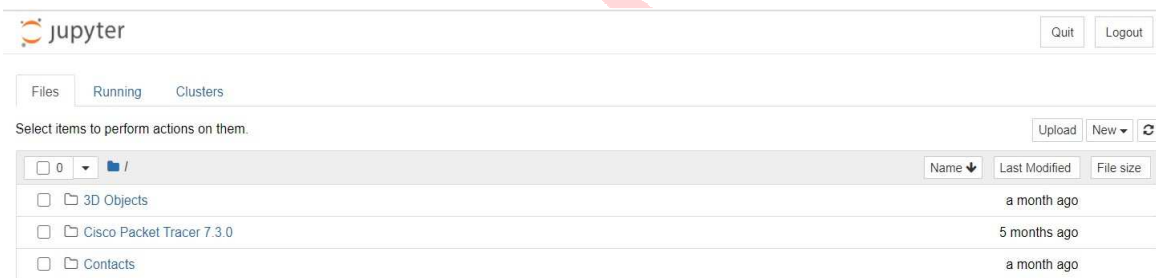
- **SciPy** – biblioteka za naučno računanje koja obuhvata funkcije za optimizaciju, linearnu algebru, procesiranje slika itd.;
- **scikit-learn** – najpopularnija biblioteka za mašinsko učenje.

U ovom kursu proći ćemo kroz sve ove biblioteke, sa akcentom na **pandas**, na kojoj se baziraju sve ostale. Shodno tome, kroz pandas ćemo objasniti i celokupan proces pripreme podataka za analizu. Za prikaz rada kroz biblioteke koristićemo web aplikaciju Jupyter Notebook.

## Jupyter Notebook

**Jupyter Notebook** je web aplikacija otvorenog koda koja se koristi za kreiranje i deljenje dokumenata, koja sadrže kod, jednačinu, vizualizaciju i tekst u realnom vremenu. Naziv *Jupyter* potiče od osnovnih programskih jezika koje podržava: *Julia*, *Python* i *R*. Ova aplikacija nije uključena u Python programski jezik. Stoga je za njeno korišćenje neophodna instalacija, koju ćemo izvršiti pozivanjem `pip install jupyter` u komandnoj liniji.

Sada kada smo instalirali Jupyter, za njegovo pokretanje je potrebno da napravimo folder, što takođe vršimo u komandnoj liniji. Pri kreiranju foldera bitno je obratiti pažnju na to u kom se direktorijumu nalazite. Preporučljivo je da to bude neki direktorijum za skladištenje dokumenata, kao što je, recimo, *Documents*. Kada se nalazimo u željenom direktorijumu, komandom `jupyter notebook` u komandnoj liniji vršimo konekciju na lokalni server ka linku <http://localhost:8888/tree>, koji se otvara u našem browseru.



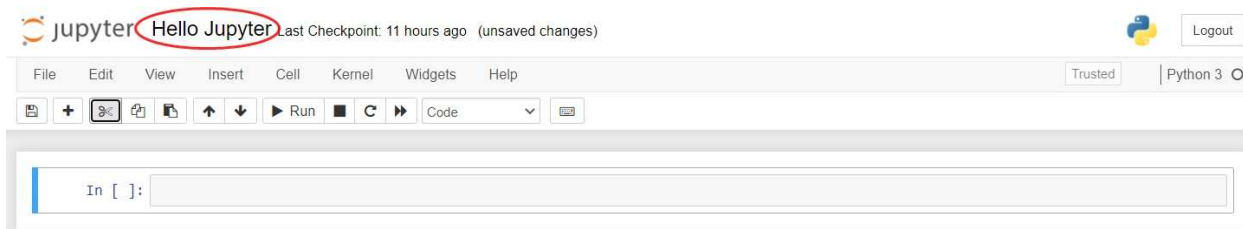
Slika 4.1. Prikaz Jupyter Notebook web aplikacije na našem lokalnom serveru

Na slici 4.1. se može primetiti da na našem serveru već postoje neki Notebook dokumenti. Ove dokumente je moguće kreirati pomoću komande *New*, koja se nalazi u desnom uglu iznad tabele sa postojećim dokumentima. Kada kliknemo na opciju *New*, kao izbor ćemo označiti *Python 3*. Ovo znači da će ćelije našeg dokumenta koristiti *Python 3* kernel, odnosno da će se kod ispisati u ćelijama čitati na tom programskom jeziku.

### Napomena

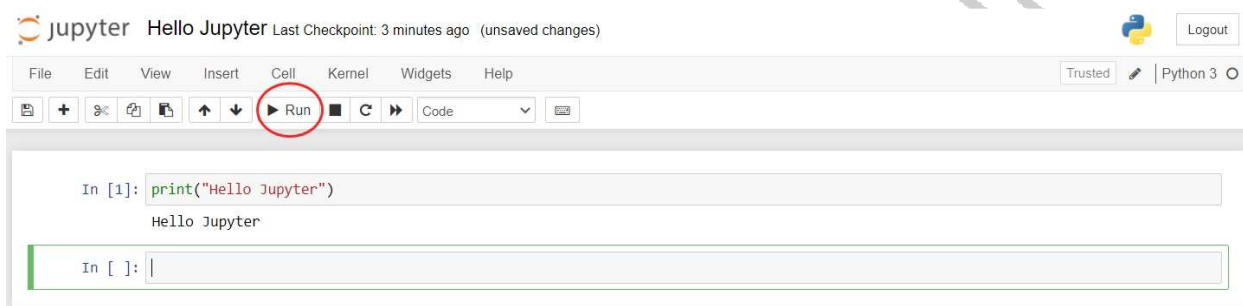
*Kernel predstavlja srce operativnog sistema, odnosno softver koji upravlja pristupom korisničkim programima, hardveru i softveru sistema.*

Novokreirani dokument inicijalno nosi ime *Untitled*. Radi preciznijeg definisanja dokumenta izmenićemo mu ime kroz klik na naslov i postaviti novi naziv, recimo *Hello Jupyter* (slika 4.2).



Slika 4.2. Promena naziva Jupyter Notebook dokumenta

Kako bismo testirali svoja podešavanja, u ćeliju ćemo umetnuti liniju koda napisanu u Python 3 programskom jeziku i pokrenuti je korišćenjem komande *Run* (*Shift+Enter*). Ispravan kod će na ekranu dati rezultat nakon pokretanja (slika 4.3).



Slika 4.3. Prikaz testiranja ćelija podešenih na Python 3 kernel

Ćelije u Notebooku su numerisane i pokreću se u tom redosledu. Imaju mogućnost deljenja varijabli i importovanih dokumenata, što omogućava deljenje koda u logičke celine bez višestrukog kreiranja. U ćelijama se može vršiti i stilizovanje teksta.

Radi lakšeg snalaženja u programu, u kratkim crtama ćemo prikazati mogućnosti koje nudi meni ovog programa:

- **File** – kreiranje novog dokumenta ili pokretanje postojećeg;
- **Edit** – uređivanje ćelija (isecanje, kopiranje, umetanje, spajanje, brisanje, razdvajanje);
- **View** – podešavanje interfejsa programa (*header*, *toolbar*, numerisanje linija koda);
- **Insert** – umetanje ćelija;
- **Cell** – pokretanje jedne ćelije, grupe ćelija ili svih ćelija, promena tipa ćelije, brisanje outputa;
- **Kernel** – promena kernela, resetovanje, rekonekcija, gašenje;
- **Widgets** – dodavanje dinamičnosti kodu pomoću JavaScripta;
- **Help** – pomoć u vezi sa korišćenjem programa.

Osim dokumenata, Jupyter Notebook dozvoljava i pokretanje tekstualnih fajlova, foldera i terminala. Takođe podržava i karticu *Running* i *Clusters* u kojima možemo da pratimo koja dokumenta i terminali su trenutno aktivni.

## Biblioteka pandas

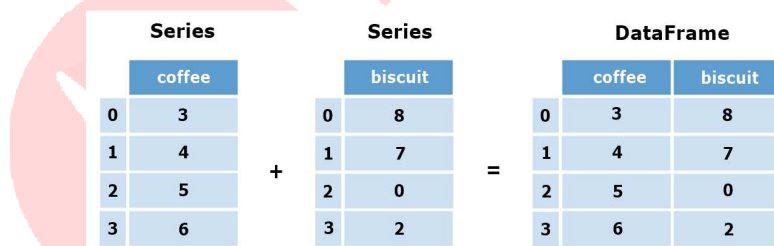
Biblioteka **pandas** predstavlja najvažnije sredstvo na raspolaganju današnjim naučnicima i analitičarima koji rade u Pythonu. Iako danas postoje mnogo moćniji alati za vizualizaciju i mašinsko učenje, pandas biblioteka predstavlja okosnicu većine projekata kad je reč o radu sa podacima. Ova biblioteka je dobila naziv od reči *panel* i *data* (*panel podaci*), što predstavlja ekonometrijski termin za skupove podataka koji uključuju zapažanja za iste podatke kroz više vremenskih perioda.

Biblioteka pandas ne samo da je centralna komponenta alata za nauku o podacima već se koristi u kombinaciji sa drugim bibliotekama u toj kolekciji. Izgrađena je na vrhu NumPy paketa, što znači da se mnoge strukture NumPyja koriste ili repliciraju u pandasu. Podaci u pandasu često se koriste i za prikaz statističkih analiza u SciPyju, crtanje funkcija iz Matplotliba i algoritme mašinskog učenja u scikit-learnu.

Jupyter Notebooks nudi dobro okruženje za korišćenje pandas biblioteke za istraživanje i modeliranje podataka, mada se ova biblioteka može koristiti i u običnim uređivačima teksta. Jupyter Notebooks nam daje mogućnost izvršenja koda u određenoj ćeliji umesto pokretanja cele datoteke. Ovo štedi puno vremena pri radu sa velikim skupovima podataka i složenim transformacijama. Takođe pruža jednostavan način za vizuelno predstavljanje pandas okvira podataka i grafikona.

Da bismo koristili pandas biblioteku, potrebno je da je prvo instaliramo. To činimo pomoću izjave `pip install pandas` u komandnoj liniji. Nakon instalacije, uvešćemo ovu biblioteku u svoj Jupyter Notebook komandom `import pandas as pd`. Sada, kada na raspolaganju imamo sadržaj pandas biblioteke, pogledajmo neke od njenih osnovnih funkcionalnosti.

Dve osnovne komponente *pandas* biblioteke su redovi (*Series*) i okviri podataka (*DataFrames*). Nad ovim komponentama je moguće vršiti identičan skup operacija kao što su popunjavanje *null* vrednosti i kalkulacija srednje vrednosti. U kakvom su odnosu ove komponente prikazano je na sledećoj fotografiji (slici 4.4).



Series			Series			DataFrame		
	coffee			biscuit			coffee	biscuit
0	3	+	0	8	=	0	3	8
1	4		1	7		1	4	7
2	5		2	0		2	5	0
3	6		3	2		3	6	2

Slika 4.4. Prikaz strukture osnovnih komponenti pandas biblioteke

Postoji više načina za kreiranje okvira podataka u pandas biblioteci, a najjednostavniji način je korišćenje rečnika podataka. Zamislimo da su posmatrani redovi zapravo raspoloživi stolovi u kafeteriji i da kolone *coffee* i *biscuit* predstavljaju broj ispijenih kafa i pojeđenih biskvita za zadatim stolom. Kako bismo porudžbine grupisali po stolovima, uvešćemo promenljivu **tables** u koju ćemo skladištiti naš okvir podataka korišćenjem jednostavnog konstruktora biblioteke `pandas pd.DataFrame()`. Ovaj konstruktor kao parametar prima naš rečnik *table*, koji ćemo pre pozivanja ovog konstruktora definisati. Za prikaz rezultata ovog okvira podatka dovoljno je da u liniji koda ispišemo samo **tables**.

**Primer kreiranja DataFrame okvira podataka u pandas biblioteci:**

```
import pandas as pd

table = {
    'coffee': [3, 4, 5, 6],
    'biscuit': [8, 7, 0, 2]
}

tables = pd.DataFrame(table)

tables
```

Out[4]:

	coffee	biscuit
0	3	8
1	4	7
2	5	0
3	6	2

Slika 4.5. Rezultat pozivanja DataFrame() konstruktora u Jupyter Notebook okruženju

Kao što je moguće primetiti na slici 4.5, DataFrame() konstruktor spaja podatke iz dve tabele korišćenjem ključ-vrednost metode. Ključevi su u ovom slučaju automatski dodeljeni brojevi u rasponu od 0 do 3. Ovi ključevi mogu biti definisani i od strane korisnika, što je u nastavku i prikazano.

**Primer ručnog dodeljivanja indeksa DataFrame okviru podataka u pandas biblioteci:**

```
import pandas as pd

table = {
    'coffee': [3, 4, 5, 6],
    'biscuit': [8, 7, 0, 2]
}

tables = pd.DataFrame(table,
    index=['Table A', 'Table B', 'Table C', 'Table D'])

tables
```

Out[6]:

	coffee	biscuit
Table A	3	8
Table B	4	7
Table C	5	0
Table D	6	2

Slika 4.6. Rezultat ručnog dodeljivanja indeksa u okviru `DataFrames()` konstruktora u Jupyter Notebook okruženju

Ovako grupisane porudžbine je sada lakše pretraživati. Funkcija pretraživanja koju pandas koristi je `loc[]`; njena uloga je vraćanje vrednosti podatka za zadati ključ. Ako bismo, na primer, želeli da saznamo šta su poručili gosti za stolom B, to možemo postići jednostavnom linijom koda `tables.loc['Table B']`.

Out[7]: coffee 4  
biscuit 7  
Name: Table B, dtype: int64

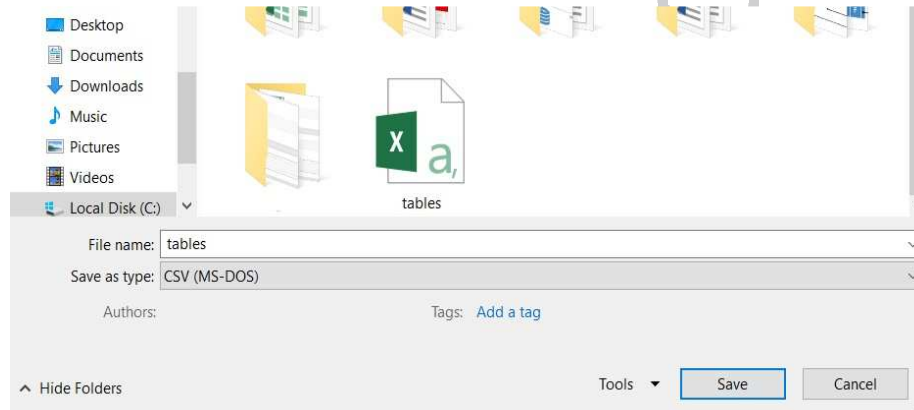
Slika 4.7. Rezultat pretraživanja okvira podataka korišćenjem `loc[]` funkcije u Jupyter Notebook okruženju

Pandas biblioteka ima mogućnost čitanja podataka sa CSV, JSON i SQL fajlova, kao i mogućnost konverzije tih podataka nazad u te fajlove. Čitanje ovih fajlova zasniva se na istom principu – kroz pokretanje funkcija namenjenih čitanju tog specifičnog fajla. Radi jednostavnosti, prikazaćemo kako to detaljnije izgleda na primeru CSV fajla.

Pre svega ćemo kreirati CSV fajl iz kog ćemo moći da povlačimo podatke. Otvorićemo novi Microsoft Excel fajl, u njega upisati okvir podataka iz našeg primera kao na prvoj slici (slika 4.8), a zatim ga sačuvati u formatu CSV kao što je to prikazano na slici posle (slika 4.9). Ovaj CSV fajl ćemo sačuvati pod imenom *tables*, jer ćemo ga kasnije pod tim imenom i čitati.

	A	B	C	D	E	F
1		coffee	biscuit			
2	Table A	3	8			
3	Table B	4	7			
4	Table C	5	0			
5	Table D	6	2			
6						
7						

Slika 4.8. Priprema okvira podataka tables za čuvanje u CSV formatu



Slika 4.9. Čuvanje tables okvira podataka u formatu CSV fajla

Ovako kreiran fajl ćemo zatim uvesti u naš Jupyter Notebook upotrebom dugmeta *Upload*, čime smo izvršili sve pripreme za čitanje podataka sa ovog fajla. S obzirom na to da CSV fajlovi ne sadrže kolonu indeksa, potrebno je da, pri pozivanju funkcije čitanja ovog fajla, podesimo vrednost kolone indeks 0. U nastavku prikazujemo komandu koja će pročitati naš CSV fajl. Ova komanda daje rezultat kao na slici 4.6.

#### Primer čitanja podataka iz tables.csv fajla:

```
csv_data = pd.read_csv('tables.csv', index_col=0)
csv_data
```

Čitanje podataka napisanih u JSON formatu vrši se na sličan način, što se jasno vidi iz sledećeg primera.

#### Primer čitanja podataka iz tables.json fajla:

```
json_data = pd.read_json('tables.json')  
  
json_data
```

Kada podatke čitamo iz baze, situacija je nešto drugačija. Ovde je neophodno da prvo uspostavimo konekciju sa bazom, da bismo kasnije tu konekciju prosledili kao parametar našoj funkciji čitanja, zajedno sa SQL upitom kao njenim prvim parametrom.

#### Primer čitanja podataka iz tables.db fajla:

```
import sqlite3  
  
con = sqlite3.connect('tables.db')  
sql_data = pd.read_sql_query(SELECT * FROM tables, con)  
  
sql_data
```

Kada smo završili sa radom nad svojim podacima, možemo ih ponovo sačuvati u željeni format. U daljem primeru je prikazano kako čuvamo podatke, odnosno, kako ih upisujemo u formatu koji želimo. Pokretanjem dole navedenih funkcija kreiraju se novi fajlovi, dok se kod baza podataka kreira u potpunosti nova tabela.

#### Primer upisivanja podataka u željenom formatu (CSV, JSON, SQL):

```
csv_data.to_csv('new_tables.csv')  
json_data.to_json('new_tables.json')  
sql_data.to_sql('new_tables', con)
```

Biblioteka pandas ima neverovatan broj mogućnosti. Da bismo mogli da pokrijemo većinu, trebaće nam veća baza raznovrsnijih podataka. Sa [ovog linka](#) preuzemo gotovu bazu podataka Netflix filmova i TV serija kreiranu u CSV formatu, a zatim je uploadovati na naš Jupyter Notebook. Fajl čitamo na isti način kao i do sada.

#### Primer čitanja netflix\_titles.csv fajla preuzetog sa kaggle.com:

```
import pandas as pd  
  
titles = pd.read_csv('netflix_titles.csv', index_col=0)
```

Nakon što smo obavili prvi korak, odnosno prikupljanje podataka, od izuzetne je važnosti da imamo vizuelni pregled kako naši podaci izgledaju. Najjednostavnije je pozivanje funkcija `head()` ili `tail()`, koje nam daju uvid u prvih, odnosno poslednjih nekoliko zapisa. Ovim funkcijama je takođe moguće proslediti parametar u vidu broja zapisa koje želimo da prikazemo, kao što ćemo mi to uraditi komandom `titles.head(3)`.



Out[63]:

	type	title	director	cast	country	date_added	release_year	rating	duration	listed_in	description
show_id											
81145628	Movie	Norm of the North: King Sized Adventure	Richard Finn, Tim Maltby	Alan Marriott, Andrew Toth, Brian Dobson, Cole...	United States, India, South Korea, China	September 9, 2019	2019	TV-PG	90 min	Children & Family Movies, Comedies	Before planning an awesome wedding for his gra...
80117401	Movie	Jandino: Whatever It Takes	NaN	Jandino Asporaat	United Kingdom	September 9, 2016	2016	TV-MA	94 min	Stand-Up Comedy	Jandino Asporaat riffs on the challenges of ra...
70234439	TV Show	Transformers Prime	NaN	Peter Cullen, Sumalee Montano, Frank Welker, J...	United States	September 8, 2018	2013	TV-Y7-FV	1 Season	Kids' TV	With the help of three human allies, the Autob...

Slika 4.10. Prikaz prva tri zapisa `netflix_titles.csv` fajla korišćenjem `head(3)` funkcije u Jupyter Notebook okruženju

Za brz uvid u osnovne podatke o strukturi baze podataka i tipovima podataka sa kojim radimo koristimo funkciju `info()`. Dakle, u našem slučaju, pozivanjem `titles.info()` dobićemo informacije prikazane na slici 4.11.

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 6234 entries, 81145628 to 70153404
Data columns (total 11 columns):
#   Column          Non-Null Count  Dtype
---  -
0   type            6234 non-null   object
1   title           6234 non-null   object
2   director        4265 non-null   object
3   cast            5664 non-null   object
4   country         5758 non-null   object
5   date_added      6223 non-null   object
6   release_year    6234 non-null   int64
7   rating          6224 non-null   object
8   duration        6234 non-null   object
9   listed_in       6234 non-null   object
10  description      6234 non-null   object
dtypes: int64(1), object(10)
memory usage: 584.4+ KB
```

Slika 4.11. Prikaz strukture podataka `netflix_titles.csv` fajla korišćenjem `info()` funkcije u Jupyter Notebook okruženju

Još jedna jednostavna funkcija je funkcija `shape`. Njenim pozivanjem kao rezultat dobijamo broj redova i kolona u vidu n-torke. U našem slučaju, `titles.shape` daće rezultat `(6234, 11)`.

Sledeći korak je eliminisanje svih duplikata iz baze. Konkretno, u našoj bazi trenutno se nalazi jedan duplikat. Kada u bazi imamo ponovljene zapise, to utiče na relevantnost podataka koji se predstavljaju kao krajnji rezultat neke analize. Iz tog razloga je neophodno rešiti se duplikata. Biblioteka `pandas` to omogućava funkcijom `drop_duplicates()`. Nakon izvršenja naredbe `titles.drop_duplicates()` pokrenućemo i komandu `titles.shape` kako bismo potvrdili da je rezultat sada `(6233, 11)`, odnosno 6233 reda i 11 kolona.

Kao i duplikate, jednako je nepoželjno imati i nepostojeće vrednosti u tabeli, jer su takve vrednosti ignorisane od strane svih operacija kao što su sumiranje, prosek, prebrojavanje i slično. Ukoliko bismo pristupili obrađivanju takvih podataka, naši rezultati bi izgubili na relevantnosti. Ovakve vrednosti se u pandas biblioteci javljaju kao `None` i `np.nan` vrednosti. Postoje dve metode rešavanja ovog problema:

- brisanje redova koji sadrže nepostojeće vrednosti;
- zamena nepostojećih vrednosti nekim drugim.

Funkcija koja nam olakšava prebrojavanje ovih vrednosti je `isnull()` funkcija, koju ćemo za našu bazu podataka pozvati komandom `titles.isnull()`. Na slici ispod je prikazano kako ova funkcija postavlja vrednosti ćelija na `True` i `False` u odnosu na to da li u ćeliji vrednost postoji ili ne (slika 4.12).

Out[75]:

	type	title	director	cast	country	date_added	release_year	rating	duration	listed_in	description
show_id											
81145628	False	False	False	False	False	False	False	False	False	False	False
80117401	False	False	True	False	False	False	False	False	False	False	False
70234439	False	False	True	False	False	False	False	False	False	False	False
80068654	False	False	True	False	False	False	False	False	False	False	False
80125979	False	False	False	False	False	False	False	False	False	False	False
...	...	...	...	...	...	...	...	...	...	...	...
80000063	False	False	True	False	False	True	False	False	False	False	False
70286564	False	False	True	False	False	True	False	False	False	False	False
80116008	False	False	True	True	True	True	False	True	False	False	False
70281022	False	False	True	False	False	True	False	False	False	False	False
70153404	False	False	True	False	False	True	False	False	False	False	False

6234 rows x 11 columns

Slika 4.12. Rezultat korišćenja `isnull()` funkcije nad `netflix_titles.csv` fajlom u Jupyter Notebook okruženju

Ova funkcija vizuelno ne može da obuhvati sve ćelije sa nepostojećim vrednostima i gotovo je nemoguće takve vrednosti prebrojati. Iz ovog razloga uvedena je funkcija `isnull().sum()`, koja rešava ovaj problem tako što sve nepostojeće vrednosti prikazuje zbirno po kolonama. Pokretanje komande `titles.isnull().sum()` daje rezultate prikazane na slici 4.13).

```
Out[76]: type           0
         title          0
         director      1969
         cast          570
         country       476
         date_added     11
         release_year   0
         rating         10
         duration       0
         listed_in     0
         description    0
         dtype: int64
```

Slika 4.13. Prikaz sumiranih nepostojećih vrednosti po kolonama korišćenjem `isnull().sum()` funkcije u Jupyter Notebook okruženju

Sledeći korak je odluka o tome da li će se nepostojeće vrednosti brisati ili će dobiti neku drugu vrednost. U slučaju da želimo da se rešimo podataka bez vrednosti, koristimo funkciju `dropna()`, odnosno za naš konkretni fajl `titles.dropna()`.

Ako bolje pogledamo količine podataka koje nedostaju, brisanje ovih zapisa dovešće do gubitka bar trećine baze podataka kojoj ti podaci koji nemaju vrednost možda i nisu od ključnog značaja. Drugi način da se nepostojeće vrednosti obrišu je da se eliminišu samo kolone koje sadrže nepostojeće vrednosti. Ovo se vrši specificiranjem parametra `axis=1` u `dropna()` funkciji.

Međutim, brisanje podataka uglavnom nije dobra praksa. Češća solucija ovakvog problema svodi se na dodeljivanje vrednosti podatku. Ova vrednost može biti **mean** ili **median**. **Mean** vrednost predstavlja prosek svih unetih podataka iz te kolone, dok **median** traži središnji element od ukupnog broja elemenata u koloni zanemarujući njegovu vrednost. Zatim se pronađena vrednost dodeljuje svim elementima koji vrednost nemaju, u našem slučaju funkcijom `titles.mean()`.

S obzirom na to da u našoj tabeli ne postoje numeričke vrednosti čiju sredinu možemo nekom dodeliti, potrebno je da tekstualne podatke dovedemo u red. Ako obratimo pažnju na sumirani izveštaj o nepostojećim vrednostima po kolonama, primetićemo da skoro trećini zapisa nedostaje podatak o režiseru.

Za početak ćemo izdvojiti kolonu režisera tako što ćemo listu režisera smestiti u promenljivu `directors`. Zatim ćemo za novokreirani okvir podataka prikazati prvih pet zapisa, kako bismo imali uvid u to kako trenutno izgledaju naši podaci u tom zapisu (slika 4.15).

#### Primer izdvajanja kolone u cilju regulisanja NA vrednosti:

```
import pandas as pd

titles = pd.read_csv('netflix_titles.csv', index_col=0)
directors = title['director']
directors.head()
```

```
Out[84]: show_id
81145628    Richard Finn, Tim Maltby
80117401                                     NaN
70234439                                     NaN
80058654                                     NaN
80125979    Fernando Lebrija
Name: director, dtype: object
```

Slika 4.14. Prikaz prvih pet zapisa liste režisera pripremljene za obradu NA podataka u Jupyter Notebook okruženju

Ono što je ostalo da bude urađeno je zamena NA vrednosti nekim konkretnim podatkom. Za zamenu koristimo funkciju `fillna()`, kojoj prosleđujemo obavezni parametar `value`, koji predstavlja novu vrednost koju želimo da postavimo; u našem slučaju će to biti **Unknown**. U vidu parametra ćemo takođe proslediti **inplace** boolean vrednost, koja za vrednost `True` ostavlja trajne izmene nad podacima. Nakon ovog izvršenja opet ćemo pokrenuti `head()` funkciju radi poređenja rezultata.

**Primer promene NA vrednosti u koloni *director* korišćenjem `fillna()` funkcije:**

```
directors.fillna(value='Unknown', inplace=True)
directors.head()
```

```
Out[87]: show_id
81145628    Richard Finn, Tim Maltby
80117401                                     Unknown
70234439                                     Unknown
80058654                                     Unknown
80125979    Fernando Lebrija
Name: director, dtype: object
```

Slika 4.15. Promena NA vrednosti podataka u koloni *director* korišćenjem `fillna()` funkcije u Jupyter Notebook okruženju

Ovaj postupak je potrebno ponavljati za sve kolone dok se ne otklone sve nepostojeće vrednosti. Bez ovog koraka nema smisla da se obrada podataka započne.

Postoje i druge pogodnosti koje nam skladištenje po kolonama nudi. Evo kako bismo, na primer, podelili zapise u kategoriju filmova i kategoriju TV serija.

**Primer kategorizacije i zbirnog prikaza zapisa prema nameni:**

```
import pandas as pd

titles = pd.read_csv('netflix_titles.csv', index_col=0)
types = titles['type']
types.value_counts()
```

```
Out[92]: Movie      4265
         TV Show   1969
         Name: type, dtype: int64
```

Slika 4.16. Rezultat kategorizacije i zbirnog prikaza zapisa korišćenjem `value_counts()` funkcije u Jupyter Notebook okruženju

Kod analize podataka često ćemo se susretati sa situacijama gde ćemo morati da isecamo, selektujemo ili ekstraktujemo deo podataka za obradu. Da bismo vršili obradu podataka, moramo znati kojeg su oni tipa, što nam omogućava funkcija `type()` koja kao parametar prima podatak i vraća njegov tip. Ako, na primer, našu promenljivu `types` definisanu u prethodnom primeru prosledimo kao parametar ovoj funkciji, odnosno izvršimo `type(types)`, za rezultat ćemo dobiti `pandas.core.series.Series`, što indikuje da je reč o koloni.

Da bismo dobili neki okvir podataka, dovoljno je da zapise neke kolone formiramo kao listu. Pogledajmo kako to izgleda na primeru formiranja okvira koji spaja kolone naziva filma `title` i godine izlaska `release_date` u promenljivoj `titles_df`.

**Primer formiranja `DataFrame` okvira podatka spajanjem kolona:**

```
import pandas as pd

titles = pd.read_csv('netflix_titles.csv', index_col=0)
titles_df = titles[['title', 'release_year']]
type(titles_df)
titles_df.head()
```

Komanda `type(titles_df)` ovoga puta za rezultat će dati **`pandas.core.frame.DataFrame`**, jer je reč o okviru podatka. Da bismo videli kako sada naši podaci izgledaju, koristimo `titles_df.head()` za prikaz prvih pet zapisa u našoj tabeli (slika 4.17).

```
Out[115]:
```

	title	release_year
show_id		
81145628	Norm of the North: King Sized Adventure	2019
80117401	Jandino: Whatever it Takes	2016
70234439	Transformers Prime	2013
80058654	Transformers: Robots in Disguise	2016
80125979	#realityhigh	2017

Slika 4.17. Prikaz `DataFrame` okvira podataka spajanjem kolona `title` i `release_year` u vidu liste u Jupyter Notebook okruženju

Za rad sa delovima liste možemo se poslužiti isecanjem podataka. Recimo da želimo da uzmemo prve tri vrednosti iz gore prikazane tabele (slika 4.17). Funkcija `iloc()` nam to omogućava tako što za parametar uzima indekse zapisa iz tabele i vraća isečeni deo tabele. Pogledajmo kako na primeru naše novoformirane tabele izgleda isecanje prva tri zapisa.

**Primer isecanja dela liste podataka korišćenjem `iloc()` funkcije:**

```
import pandas as pd

titles = pd.read_csv('netflix_titles.csv', index_col=0)
titles_df = titles[['title', 'release_year']]
sliced_titles = titles_df.iloc[0:3]
sliced_titles
```

Out[117]:

	title	release_year
show_id		
81145628	Norm of the North: King Sized Adventure	2019
80117401	Jandino: Whatever it Takes	2016
70234439	Transformers Prime	2013

Slika 4.18. Isecanje liste podataka korišćenjem `iloc()` funkcije u Jupyter Notebook okruženju

Za selekciju dela podataka moguće je koristiti i uslove. U sledećem primeru prikazaćemo kako da iz tabele koja sadrži preko 6000 filmova i TV serija možemo da selektujemo samo one koje su režirali Christopher Nolan i Quentin Tarantino.

**Primer korišćenja uslova za selekciju podataka iz liste:**

```
import pandas as pd

titles = pd.read_csv('netflix_titles.csv', index_col=0)

titles_df = titles[['director', 'title', 'release_year']]

titles_df[(titles_df['director']=='Christopher Nolan') |
           (titles_df['director']=='Quentin Tarantino')].head()
```

Ovaj uslov je mogao da se napiše i na sledeći način, a da rezultat ostane nepromenjen:

```
titles_df[titles_df['director'].isin(['Christopher Nolan', 'Quentin
Tarantino'])].head()
```



Out[121]:

	director	title	release_year
show_id			
80064515	Quentin Tarantino	The Hateful Eight	2015
70108777	Quentin Tarantino	Inglourious Basterds	2009
70131314	Christopher Nolan	Inception	2010
60031236	Quentin Tarantino	Kill Bill: Vol. 1	2003
60032563	Quentin Tarantino	Kill Bill: Vol. 2	2004

Slika 4.19. Prikaz selekcije podataka korišćenjem uslova u Jupyter Notebook okruženju

Na prikaz podataka u pandas biblioteci mogu se primeniti i funkcije. Ono što je potrebno je da definišemo funkciju koja će da selektuje podatke na način na koji mi želimo, a zatim je pomoću `apply()` funkcije pandas paketa primenimo na željeni podatak.

Na našem primeru, prikazano je kako funkciju za kategorizaciju `category_function()` primenjujemo nad podacima `release_year`. Funkcija koju nad ovim podacima primenjujemo ponaša se na sledeći način: uzima podatak iz kolone i njegovu vrednost postavlja za parametar, da bi se tako prosleđen parametar poredio sa vrednošću u funkciji. Ovaj postupak se ponavlja za svaki podatak u listi nad kojom obavljam funkciju. U nastavku prikazujemo kako rezultati ove funkcije izgledaju u prva tri zapisa naše tabele.

#### Primer korišćenja funkcije za selekciju podataka:

```
import pandas as pd

titles = pd.read_csv('netflix_titles.csv', index_col=0)

def category_function(x):
    if x >= 2015:
        return "new movie"
    else:
        return "old movie"

df = titles[['title', 'release_year']]
df['release_category'] = df['release_year'].apply(category_function)
df.head(3)
```

Out[135]:

	title	release_year	release_category
show_id			
81145628	Norm of the North: King Sized Adventure	2019	new movie
80117401	Jandino: Whatever it Takes	2016	new movie
70234439	Transformers Prime	2013	old movie

Slika 4.20. Prikaz rezultata primene korisnički definisane funkcije `category_function()` nad listom podataka u Jupyter Notebook okruženju

## Biblioteka Matplotlib

**Matplotlib** je jedna od najpopularnijih Python biblioteka za vizualizaciju podataka. Ova biblioteka je višepatformska, odnosno, nije Python-specifična. Služi za kreiranje 2D skica na osnovu podataka skladištenih u nizovima, zbog čega koristi NumPy numeričku biblioteku Pythona, o kojoj će biti više reči u nastavku.

Za instalaciju ove biblioteke koristimo `pip install matplotlib` izjavu u komandnoj liniji, kao što smo to činili za ostale instalacije. Preduslovi za korišćenje ove biblioteke su posedovanje Python verzije 2.7 ili novije, kao i posedovanje NumPy biblioteke, koja dolazi zajedno sa instalacijom Python programskog jezika. U nastavku ćemo prikazati kako se vizualizacija koristi u Pythonu na primeru jednog jednostavnog nacrt.

Pre početka rada neophodno je da uvedemo Matplotlib biblioteku pod pseudonomom `plt` kao konvencijom pisanja. Zatim ćemo uvesti NumPy biblioteku, pod pseudonimom `np`, jer nam je potreban niz brojeva za pravljenje nacrt.

Za uvođenje n-dimenzionalnog niza koristimo funkciju `arange()` NumPy biblioteke, čija prva dva parametra predstavljaju raspon niza, a treći povećanja vrednosti u tom rasponu, kao što to čini funkcija `range()` koju smo obradili u jednoj od prethodnih lekcija. Ovako raspoređenim elementima po x-osi dodaćemo sinusne vrednosti uglova pomoću funkcije `sin()` i dodeliti ih y-osi za realniji prikaz na našem grafiku. Za skiciranje našeg grafika koristimo `plot()` funkciju `matplotlib` biblioteke. Za iscrtavanje grafika moguće je definisati nazive x i y ose, kao i naziv krive funkcije koja je definisana. Sa ovako definisanim podacima moguće je izvršiti prikaz, što činimo komandom `show()`, koja za ulogu ima pozivanje prozora koji će ovaj nacrt grafički prikazati. Pogledajmo kako to izgleda u kodu.

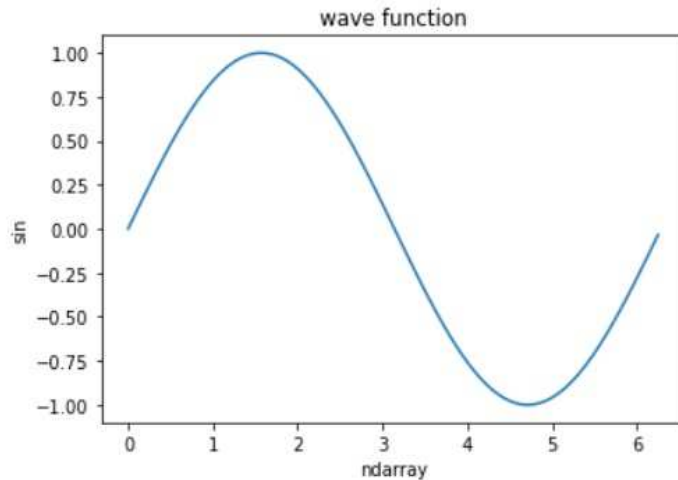
### Primer kreiranja jednostavnog nacrt korišćenjem biblioteka Matplotlib i NumPy:

```
from matplotlib import pyplot as plt
import numpy as np
import math

x = np.arange(0, math.pi*2, 0.05)
y = np.sin(x)

plt.plot(x,y)
plt.xlabel("ndarray")
plt.ylabel("sin")
plt.title("wave function")
plt.show()
```





*Slika 4.21. Prikaz grafika kreiranog korišćenjem Matplotlib biblioteke u Jupyter Notebook okruženju*

Postoji više načina za vizualizaciju podataka, kao što su, na primer, histogram ili *pie* dijagram. Za dvodimenzionalni prikaz nacrta neophodno je da u programu definišemo barem jednu regiju za prikaz i barem jednu osovinu koja sadrži dva objekta (x i y osu) kada je reč o 2D prikazu. 3D prikaz mora imati osovinu sa minimum tri objekta, odnosno tri ose.

Regija za prikaz u biblioteci Matplotlib je sadržana u klasi `Figure`, koja predstavlja natklasu svih klasa nacrta. Svaka regija za prikaz može imati više osovinu, dok jedna osovina može biti sadržana samo u jednoj regiji. Objekat osovine ili `Axes` je osovinska regija koja služi za prikaz podataka. Pri kreiranju dvodimenzionalnih nacrta, neophodno je definisati podnacrte koji sadrže i regiju za prikaz i osovinski objekat, što činimo na sledeći način:

```
fig,ax = plt.subplot(subplot(nrows, ncols, index))
```

Da bismo materiju približili i povezali je sa dosadašnjim znanjem iz `pandas` biblioteke, prikazaćemo kako Matplotlib može da posluži na našem primeru Netflix liste filmova i TV serija. Pogledajmo kako bi izgledao jedan histogram broja zapisa po godinama izlaska.

#### **Primer korišćenja Matplotlib biblioteke za prikaz histograma podataka:**

```
import pandas as pd
from matplotlib import pyplot as plt
import numpy as np

titles = pd.read_csv('netflix_titles.csv', index_col=0)
values = titles['release_year']

fig,ax = plt.subplots(1,1)

ax.hist(values, bins=np.arange(2010,2021,1))

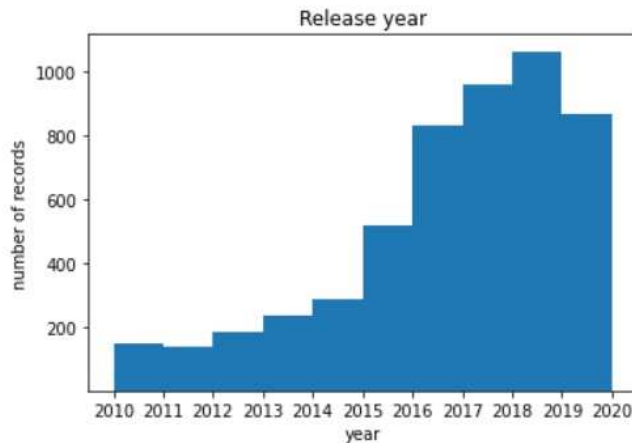
ax.set_title("Release year")

ax.set_xticks(np.arange(2010,2021,1))
```

```
ax.set_yticks(np.arange(200, 1200, 200))

ax.set_xlabel('year')
ax.set_ylabel('number of records')

plt.show()
```



Slika 4.22. Prikaz histograma kreiranog korišćenjem Matplotlib biblioteke na primeru netflix\_titles.csv podataka u Jupyter Notebook okruženju

### Pitanje

Za učitavanje CSV fajla sa podacima koristimo funkciju:

- `pd.open_csv()`
- `pd.import_csv()`
- **`pd.read_csv()`**
- `pd.load_csv()`

### Objašnjenje:

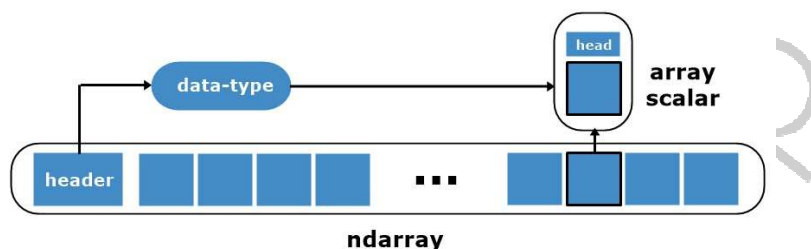
Za učitavanje fajla sa podacima koristimo `pd.read_csv()`. Ostale navedene funkcije ne postoje.

## Biblioteka NumPy

Biblioteka **NumPy** nastala je od reči *numerički Python*. Ova biblioteka se sastoji od višedimenzionalnih nizova objekata i kolekcije funkcija koje se nad tim nizovima obavljaju. Uz pomoć biblioteke NumPy možemo da vršimo matematičke i logičke operacije, manipulaciju oblika, operacije linearne algebre i generisanje slučajnih vrednosti.

Kombinacija ove biblioteke sa Matplotlibom i Scipyjem, o kojem će biti više reči u nastavku, često se koristi kao zamena za MatLab računski jezik. Instalacija ove biblioteke nije potrebna, jer ona dolazi u paketu sa instalacijom Pythona.

Najvažniji objekat u NumPy biblioteci je  $n$ -dimenzionalni niz ili `ndarray`. Ovaj niz skladišti kolekciju podataka istog tipa u  $n$  dimenzija. Stavkama ove kolekcije može se pristupiti pomoću brojeva indeksa, koji počinju od broja 0. Svaka stavka u ovom nizu ima veličinu bloka dodeljenu u memoriji. Svaki element u memoriji je objekat tipa podatka ili `dtype`. Na sledećoj slici prikazan je odnos  $n$ -dimenzionalnog niza sa objektom tipa podatka.



Slika 4.23. Odnos  $n$ -dimenzionalnog niza i objekta tipa podatak

Postoji više načina kreiranja instance klase  $n$ -dimenzionalnog niza. Osnovna komanda za izvršenje kreiranja je `numpy.array`. Ova funkcija može da primi nekoliko parametara:

```
numpy.array(object, dtype = None, copy = True,  
order = None, subok = False, ndmin = 0)
```

Prvi parametar odnosi se na objekat koji pretvaramo u niz. Ostali parametri su opcioni i odnose se na definisanje tipa podatka, pravljenje kopija, sortiranje, davanje permisija potklasama i određivanje minimalnog broja dimenzija niza, respektivno.

#### Primer kreiranja $n$ -dimenzionalnog niza korišćenjem NumPy biblioteke:

```
import numpy as np  
  
a = np.array([[0, 1], [2, 3]])
```

Objekat tipa podatka opisuje interpretaciju fiksnog bloka memorije koji odgovara polju u zavisnosti od sledećih aspekata:

- tip podatka – može biti *int*, *float* ili *Python Object*;
- veličina podatka;
- raspored bajtova – *big endian* ili *little endian*;
- ako su podaci strukturirani – imena polja, tip podatka i memorijska lokacija za svako polje;
- ako su podaci podnizovi – oblik i tip podataka.

## Napomena

*Redosled bajtova određuje se prefiksom > ili <, što indikuje kako su bajtovi u memoriji raspoređeni. Znak > koristimo za big endian, koji radi po principu skladištenja najznačajnijih bajtova na najmanjoj adresi, dok little endian koristi < prefiks za skladištenje najmanje važnih bajtova na najmanjoj adresi.*

Objekat tipa podatka koristi strukturu `numpy.dtype(object, align, copy)`, gde imamo parametar `object`, koji predstavlja tip podatka koji želimo da dodelimo; `align`, čija `True` vrednost skladišti sve na istoj memorijskoj lokaciji, i `copy`, čija `False` vrednost označava da će podaci koristiti reference predefinisanih objekata.

U nastavku ćemo prikazati primer jednog definisanog objekta tipa podatka. Ovaj objekat kao parametar prima strukturirane podatke koji obuhvataju jednu *int*, jednu *String* i jednu *float* vrednost. Objekat tipa podataka definisan je u promenljivoj *employee*.

## Primer kreiranja objekta tipa podatka korišćenjem NumPy biblioteke:

```
import numpy as np

employee = np.dtype([('id', 'i1'), ('name', 'S20'), ('salary', 'f4')])

a = np.array([(100, 'Jack', 2500.0), (101, 'Johnnie', 1500.0)],
             dtype = employee)
```

Biblioteka NumPy nudi različite mogućnosti u pogledu naprednih računskih operacija koje bez uvđenja ovog paketa nisu moguće. Tu spadaju kreiranje matrica, matematičke, statističke i aritmetičke operacije, funkcije stringova, indeksiranje i slično. Za konkretan primer upotrebe ove biblioteke možemo se osvrnuti na `numpy.arange(start, stop, step, dtype)`, čiju smo praktičnu primenu prikazali na našem primeru sa obradom *netflix\_titles.csv* podataka u kombinaciji sa Matplotlib bibliotekom (slika 4.22).

## Biblioteka SciPy

Biblioteka **SciPy** daje mogućnost naprednog računanja u oblastima matematike, nauke i inženjerstva. Ova biblioteka zavisi od NumPyja, koji pruža pogodnu i brzu manipulaciju n-dimenzionalnim nizovima. Napravljena je u svrhu poboljšanja numeričke integracije i optimizacije ovih nizova radi veće efikasnosti i lakšeg korišćenja.

Ova biblioteka dolazi zajedno sa instalacijom Python programskog jezika. Alternativna opcija je instalacija popularne pandas biblioteke koja sadrži SciPy pozivanjem sada već poznate komande u našoj komandnoj liniji: `pip install scipy`.

Uvođenje SciPy biblioteke u naš program ne zahteva eksplicitno uvođenje NumPyja. Glavni objekat NumPyja je homogeni višedimenzionalni niz koji sadrži elemente iste vrste indeksirane pozitivnim celim brojevima. Dimenzije ovih nizova označene su osama, dok broj osa predstavlja rank. Kako ovi nizovi rade sa podacima – imali smo priliku da vidimo u segmentu posvećenom NumPy biblioteci, te ovde to nećemo ponovo objašnjavati.

SciPy podržava neverovatno mnogo mogućnosti i modula koji nam omogućavaju da vršimo operacije klasteriranja, fizičkih i matematičkih ograničenja, interpolacije, ulaza i izlaza, linearne algebre, optimizacije itd. Kako ova biblioteka utiče na poboljšanje nizova sa više dimenzija – proverićemo na primeru klasteriranja.

**K-means** klastering je metoda za određivanje grupa podataka (klastera) i njihovih centara u skupu neobeležanih podataka. Klaster možemo posmatrati kao skup podataka (tačaka) čije je rastojanje malo u poređenju sa rastojanjem od tačaka izvan klastera. *K-means* ponavlja sledeća dva koraka u procesu klasteriranja:

- za svaki centar identifikuje se skup tačaka koje su bliže njemu nego bilo kom drugom centru;
- u svakom klasteru računa se srednja vrednost obeležja podataka, koja postaje novi centar.

Ovi koraci se ponavljaju dok centri ne prestanu da se pomeraju ili podaci ne prestanu da se menjaju. SciPy ima dobru implementaciju k-meansa, koju ćemo testirati na našem primeru sa podacima iz `netflix_titles.csv` fajla. Za početak ćemo uvesti `kmeans`, `vq` i `whiten` funkcije iz naše biblioteke koja radi sa klasterima, a zatim ćemo objasniti šta svaka od njih znači.

Kao podatke za rad uzećemo ranije kreiranu listu kolone `release_year` iz našeg standardnog CSV fajla o filmovima i TV serijama. Korišćenjem `whiten()` funkcije, svaki od podataka podeljen je standardnim odstupanjem kako bi se dobila varijacija po jedinici. Funkcija `kmeans()` vrši klasterizaciju elemenata po gore objašnjenom postupku, sve dok ne postigne broj klastera koji je zadat u vidu parametra zajedno sa podacima koji su prosleđeni za klasteriranje. Te klastere nazivamo centroidima. Na kraju funkcija `vq()` upoređuje svaki vektor sa centroidima i dodeljuje posmatranje najbližem klasteru. Da bismo imali predstavu o tome kako ovi podaci zapravo izgledaju, prikazaćemo ih na ekranu.

#### Primer klasteriranja podataka korišćenjem SciPy biblioteke:

```
import pandas as pd
from scipy.cluster.vq import kmeans,vq,whiten

titles = pd.read_csv('netflix_titles.csv', index_col=0)
values = titles['release_year']

values = whiten(values)
Centroids,_ = kmeans(values,5)
clx,_ = vq(values,centroids)

print(centroids)
print(clx)
```

Rezultat ispisivanja:

```
[228.69777413 229.03154631 228.08294991 223.96282228 226.87700184]
[1 0 0 ... 0 0 4]
```

## Biblioteka scikit-learn

Biblioteka **scikit-learn** je najpoznatija besplatna Python biblioteka za mašinsko učenje, koja sadrži različite algoritme klasifikacije, regresije, klasteriranja i sl. Mašinsko učenje predstavlja granu u računarskoj nauci koja proučava dizajn algoritama koji imaju sposobnost učenja. Tipični zadaci su konceptualno učenje, funkcionalno učenje ili prediktivno modeliranje, klasteriranje i pronalaženje predvidivih obrazaca. Ti se zadaci uče putem dostupnih podataka koji su promatrani kroz iskustva i uputstva.

S obzirom na to da je ova biblioteka izgrađena na SciPy biblioteci, za njeno funkcionisanje potrebno je posedovanje sledećih paketa: NumPy, SciPy, Matplotlib, iPython, SymPy, pandas. Pored ovoga, potrebno je, naravno, i da komandom `pip install sklearn-learn` pokrenemo instalaciju scikit-learn biblioteke.

Nakon instalacije, kao i sa ostalim bibliotekama, počinjemo rad uvođenjem biblioteke u program komandom `import sklearn`. Da bismo prikazali kako ova biblioteka radi sa podacima, prvo moramo da definišemo te podatke. Ova biblioteka ima modul **datasets**, koji automatski generiše neki set veštačkih podataka. U narednom primeru prikazano je kako generišemo podatke za rad putem funkcije `load_digits()`.

### Primer generisanja veštačkog seta podataka korišćenjem scikit-learn biblioteke:

```
from sklearn import datasets

digits = datasets.load_digits()
print(digits)
```

Kada pokrenete ovaj kod, primetićete da se u vašem *Jupyter Notebook* okruženju pojavio neki set podataka koji je skladišten u rečniku, što znači da u svojoj strukturi sadrži neke ključ-vrednost parove. Radi lakšeg snalaženja u ovom setu podataka, proverićemo sa kojim ključevima u tom setu raspoložemo pomoću komande `print(digits.keys())`. Zadana komanda nam prikazuje sledeće:

```
dict_keys(['data', 'target', 'frame', 'feature_names', 'target_names',
'images', 'DESCR'])
```

Ovakav pregled podataka već omogućava lakše pretraživanje. Pogledajmo, na primer, kakvu strukturu podataka sadrže *data* i *target* ključevi. Prvi ključ predstavlja posmatrane podatke koji su prikupljeni, dok drugi predstavlja ciljani skup podataka koje želimo bolje da razumemo. Njihovu strukturu ćemo proveriti pomoću već korišćene funkcije koju sadrži i *pandas* biblioteka, a to je `shape()`.

### Primer ispitivanja strukture podataka u scikit-learn biblioteci:

```
from sklearn import datasets

digits = datasets.load_digits()

digits_data = digits.data
print(digits_data.shape)
```

```
digits_target = digits.target
print(digits_target.shape)
```

Kao rezultat dobićemo (1797, 64) i (1797,). Ovo znači da su podaci skladišteni u 1797 redova i 64 kolone i da imamo 1797 ciljanih vrednosti.

Kao što smo već spomenuli, scikit-learn biblioteka je u potpunosti prilagođena mašinskom učenju. Njeni moduli su vrlo jednostavni za korišćenje, što je dodatan razlog za njihovu široku primenu. U nastavku prikazujemo konkretnu primenu ove biblioteke na primeru stabla klasifikacije i regresije.

Počnemo sa uvođenjem modula za setove podataka, metrike i drva odlučivanja. Kao set podataka koristićemo ugrađeni Pythonov set *Iris*, nad kojim ćemo primeniti algoritam drva odlučivanja CART (Classification and Regression Trees). Model podataka koji ćemo koristiti je **DecisionTreeClassifier()**. Njemu ćemo proslediti sistemski generisane podatke i ciljane vrednosti pomoću ugrađene funkcije `fit()`. Kao očekivane – *expected* podatke definisaćemo ciljane, a za predviđanje – *predicted* ćemo koristiti ugrađenu funkciju `predict()`, kojoj ćemo proslediti podatke. Na kraju štampamo rezultate tačnosti klasifikacije i matrice konfuzije.

#### Primer klasifikacije stabla i regresije korišćenjem scikit-learn biblioteke:

```
from sklearn import datasets
from sklearn import metrics
from sklearn.tree import DecisionTreeClassifier

dataset = datasets.load_iris()
model = DecisionTreeClassifier()
model.fit(dataset.data, dataset.target)

expected = dataset.target
predicted = model.predict(dataset.data)
print(metrics.classification_report(expected, predicted))
print(metrics.confusion_matrix(expected, predicted))
```

```
              precision    recall  f1-score   support

    0               1.00      1.00      1.00        50
    1               1.00      1.00      1.00        50
    2               1.00      1.00      1.00        50

 accuracy               1.00          150
 macro avg              1.00          150
 weighted avg          1.00          150

[[50  0  0]
 [ 0 50  0]
 [ 0  0 50]]
```

Slika 4.24. Rezultati tačnosti klasifikacije i matrice konfuzije u Jupyter Notebook okruženju

## Regresija

Nauka podataka i mašinsko učenje pokreću odluke u finansijskom i energetskom sektoru, napredak medicine, porast društvenih mreža i još mnogo toga. Linearna regresija jedna je od osnovnih tehnika statističkog i mašinskog učenja. Bez obzira na to da li želite da uradite statistiku, mašinsko učenje ili naučno računanje, šanse da ćete se susresti sa linearnom regresijom su velike.

Regresija traži odnose između promenljivih. Na primer, možemo pokušati da utvrdimo matematičku zavisnost cena kuća od njihovih površina, broja spavaćih soba, udaljenosti od centra grada i slično. Generalno, u regresijskoj analizi obično uzimamo u obzir neki fenomen koji nas zanima i pravimo niz zapažanja. Svako zapažanje ima dve ili više karakteristika. Još jedna od pretpostavki je da bar jedna od tih karakteristika zavisi od ostalih. Cilj regresije je da pokušamo da uspostavimo zavisnost među tim karakteristikama.

Drugim rečima, potrebna nam je funkcija koja neke karakteristike ili promenljive dovoljno dobro mapira sa drugima. Ova funkcija ima ulaze kao nezavisne promenljive koje se nazivaju još i *prediktori*, i izlaze kao zavisne promenljive, odnosno *autpute*. Nezavisne promenljive  $x$  mogu biti kontinualne, diskretne i kategoričke (pol, nacionalnost i sl.), dok zavisne promenljive  $y$  obično čini jedna varijabla koja je kontinualna. Ako postoje dve ili više nezavisnih promenljivih, one se označavaju kao vektor  $x=(x_1,...,x_n)$ , gde je  $n$  broj vektora. Linearnu regresiju prema broju izlaza možemo podeliti na jednostavnu, odnosno regresiju sa jednim izlazom, i višestruku, tj. onu koja ima više od jednog izlaza. U nastavku ćemo svoja objašnjenja temeljiti na primeru jednostavne regresije.

Ako promenljivom  $b_0$  obeležimo koeficijent korelacije ulaza i izlaza, a promenljivom  $b_1$  nagib regresione prave, formula koja određuje funkciju linearne regresije će izgledati ovako:

$$y = b_0 + b_1 * x$$

Recimo da se  $y$  odnosi na obeležje telesne težine i da želimo da ispitamo da li se ona menja sa promenama visine  $x$ . Dakle, u tom slučaju bi težina bila zavisna varijabla, a visina nezavisna. U našem primeru radićemo na takvom skupu podataka. Cilj je napraviti regresioni model koji će za rezultat imati što veći broj vrednosti koje pripadaju linearnoj pravoj i što manje vrednosti sa kvadratnim odstupanjima.

Kao set podataka za rad uzećemo javno dostupne podatke sa [ovog linka](#). Reč je o fajlu *data.csv*, koji sadrži podatke o visini i težini slučajno izabranih pojedinaca. Učitavanje fajla vršimo pomoću *pandas* biblioteke, kao što je to objašnjeno ranije u ovoj lekciji.

### Napomena

*Veoma je važno da fajl uploadujete u svoj Jupyter Notebook da bi učitavanje podataka moglo da se izvrši. U suprotnom će se pojaviti greška o nepostojećem fajlu.*

Podatke iz učitanoj fajla zatim smeštamo u promenljive  $x$  i  $y$  tako što uzimamo vrednosti sa indeksiranih lokacija pomoću funkcije `iloc[]`, koja prikuplja sve podatke iz zadate kolone. Pre nego što počnemo da izrađujemo model, podatke iz kolona  $x$  i  $y$  moramo da podelimo na **test** i **training** podatke. *Test* podaci predstavljaju porciju podataka koja se testira, dok *training* set služi za izgradnju modela, odnosno, na tim podacima prilagođavamo svoj model.



Kao parametar funkcije `train_test_split`, koja razdvaja promenljive, uvodimo dve nove promenljive, u koje smeštamo podatke i veličinu porcije podataka za testiranje.

Sledeći korak je uklapanje linearne regresije u *training* set, a zatim i predviđanje izlaza *y* na osnovu vrednosti parametra *x*, kao što smo to činili u primeru sa stablom odlučivanja korišćenjem *scikit-learn* biblioteke. Na kraju, korišćenjem *Matplotlib* biblioteke vizualizujemo dobijene podatke, kao što činili i ranije u ovoj lekciji.

#### Primer linearne regresije korišćenjem biblioteka za analizu podataka:

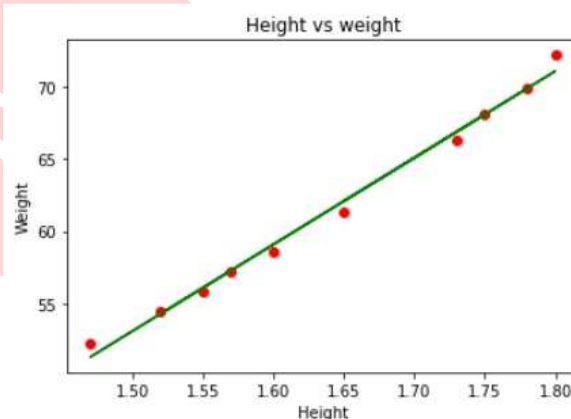
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

dataset = pd.read_csv('data.csv')
x = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size =
0.2)

model = LinearRegression()
model.fit(x_train, y_train)
y_pred = model.predict(x_test)

plt.scatter(x_train, y_train, color = 'red')
plt.plot(x_train, model.predict(x_train), color = 'green')
plt.title('Height vs weight')
plt.xlabel('Height')
plt.ylabel('Weight')
plt.show()
```



*Slika 4.25. Rezultat analize podataka korišćenjem modela Linearne Regresije u Jupyter Notebook okruženju*

## Rezime

- Analiza podataka podrazumeva proces inspekcije, filtriranja, transformacije i modeliranja podataka u cilju formiranja korisnih informacija, donošenja informisanih odluka i podrške u odlučivanju.
- Za analizu podataka možemo koristiti alate za automatsko upravljanje ili programske jezike.
- Analiza podataka ima pet ključnih faza: prikupljanje podataka, prečišćavanje, oblikovanje, proučavanje i aktivnosti nad podacima.
- Jupyter Notebook je web aplikacija otvorenog koda koja se koristi za kreiranje i deljenje dokumenata koji sadrže kod, jednačinu, vizualizaciju i tekst u realnom vremenu.
- Na biblioteci pandas temelji se struktura analiza podataka u Pythonu. Njene osnovne komponente su kolone i okviri podataka. Pruža mogućnost učitavanja CSV, JSON i SQL fajlova. Podržava funkcije pronalaženja nepostojećih vrednosti – `data.isna().sum()`, otpuštanja tih vrednosti – `data.dropna()` i njihove zamene – `data.fillna()`, kao i mnoge druge funkcije namenjene pripremi podataka za analizu, kao što je npr. otpuštanje duplikata – `pd.drop_duplicates()`.
- Matplotlib je biblioteka za vizualizaciju podataka. Koristi NumPy numeričku biblioteku. Svaki 2D nacrt u ovoj biblioteci mora da ima jednu regiju za prikaz – *Figure*, koja predstavlja natklasu biblioteke, i jednu osovinsku regiju – *Axes*, koja sadrži dva objekta (x i y osu). Nacrt definišemo linijom `fig,ax=plt.subplot(nrows, ncols)`.
- NumPy je numerička biblioteka na kojoj se baziraju n-dimenzionalni nizovi – *ndarray*. Ovi nizovi skladište kolekciju podataka istog tipa u n dimenzija. Svaka stavka u ovom nizu ima veličinu bloka dodeljenu u memoriji, a svaki element u memoriji predstavlja objekat tipa podatka ili *dtype*.
- SciPy je biblioteka naprednog računanja u oblastima matematike, nauke i inženjerstva. Zahteva podršku NumPy biblioteke za manipulaciju n-dimenzionalnim nizovima. Sadrži module za operacije klasteriranja, fizičkih i matematičkih ograničenja, interpolacije, ulaza i izlaza, linearne algebre, optimizacije itd.
- Najpopularija biblioteka za mašinsko učenje je scikit-learn. Sadrži različite algoritme klasifikacije, regresije, klasteriranja i sl. Posедуje ugrađeni modul za veštačko generisanje podataka *dataset*. Koristi funkciju `fit()` za kreiranje modela sa željenim parametrima i funkciju `predict()` za predviđanje.
- Regresija je ispitivanje zavisnosti izlaznih komponenti od ulaznih. Regresija može biti jednostavna (sa jednim izlazom) i višestruka (sa više izlaza). Cilj je generisanje što manjih vrednosti kvadratnih odstupanja. Koristi scikit-learn, NumPy i Matplotlib kao obavezne biblioteke. Primenjuje train/test metodu, koja odvaja podatke na dve porcije: za izgradnju modela i za testiranje tačnosti modela.